

Anvaya: An Algorithm and Case-Study on Improving the Goodness of Software Process Models generated by Mining Event-Log Data in Issue Tracking Systems

Perna Juneja*, Divya Kundra* and Ashish Sureka†

*Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

perna1399@iiitd.ac.in, divya1395@iiitd.ac.in

†Software Analytics Research Lab (SARL), India

ashish@iiitd.ac.in

Abstract—Issue Tracking Systems (ITS) such as Bugzilla can be viewed as Process Aware Information Systems (PAIS) generating event-logs during the life-cycle of a bug report. Process Mining consists of mining event logs generated from PAIS for process model discovery, conformance and enhancement. We apply process map discovery techniques to mine event trace data generated from ITS of open source Firefox browser project to generate and study process models. Bug life-cycle consists of diversity and variance. Therefore, the process models generated from the event-logs are spaghetti-like with large number of edges, inter-connections and nodes. Such models are complex to analyse and difficult to comprehend by a process analyst. We improve the Goodness (fitness and structural complexity) of the process models by splitting the event-log into homogeneous subsets by clustering structurally similar traces. We adapt the K-Medoid clustering algorithm with two different distance metrics: Longest Common Subsequence (LCS) and Dynamic Time Warping (DTW). We evaluate the goodness of the process models generated from the clusters using complexity and fitness metrics. We study back-forth & self-loops, bug reopening, and bottleneck in the clusters obtained and show that clustering enables better analysis. We also propose an algorithm to automate the clustering process - the algorithm takes as input the event log and returns the best cluster set.

Index Terms—Bug Tracking System, Clustering, Mining Software Repositories, Process Mining, Process Model Fitness Metric, Process Model Structural Complexity

I. RESEARCH MOTIVATION AND AIM

Software Process Intelligence (SPI) is an emerging and evolving discipline involving mining and analysis of software processes. This is modeled on the lines of application of Business Intelligence techniques to business processes (Business Process Intelligence (BPI)), but with the focus on software processes and its applicability to Software Engineering (SE) and Information Technology (IT) systems. SPI has diverse applications and is an area that has recently attracted several researcher's attention due to availability of vast data generated during software development. Some of the business applications of process mining on software repositories or SPI are: uncovering runtime process models, discovering process inefficiencies and inconsistencies, observing project key indi-

cators and computing correlation between product and process metrics, extracting general visual process patterns for effort estimation and analyzing problem resolution activities [1] [2].

Several SE processes such as issue or defect resolution are flexible and consists of several process variants (that are adhoc and unstructured) and a wide spectrum of behavior. This results in a spaghetti process model consisting of a large number of activity or task nodes as well as a large number of relations (or directed edges) between these nodes. A spaghetti process model is structurally complex and hard to comprehend for a process analyst. Trace clustering is a technique which has been applied on business process logs to split a given event-log into homogenous subsets from which process models are uncovered. Trace clustering has shown to improve the comprehensibility of process models in environments which allow process flexibility and large number of variants. The research motivation of the study presented in this paper is to investigate the application of trace clustering in the domain of SPI and process mining software repositories. The specific research aim of the work presented in this paper are the following:

- 1) To study the problem of spaghetti process models in the domain of software defect and issue resolution by conducting a case-study on open-source Firefox browser project.
- 2) To propose a trace clustering technique based on grouping sequential data and apply it on issue tracking system dataset of a large, complex and log-lived open-source project. To investigate the effectiveness of the proposed trace clustering technique in reducing the structural complexity and enhancing the process model comprehensibility for a process analyst.
- 3) To study self-loops, back-and-forth, issue reopen and bottlenecks on the discovered process models from the homogeneous subset output of trace clustering and illustrate its benefits in the domain of SPI using a real-life case-study.

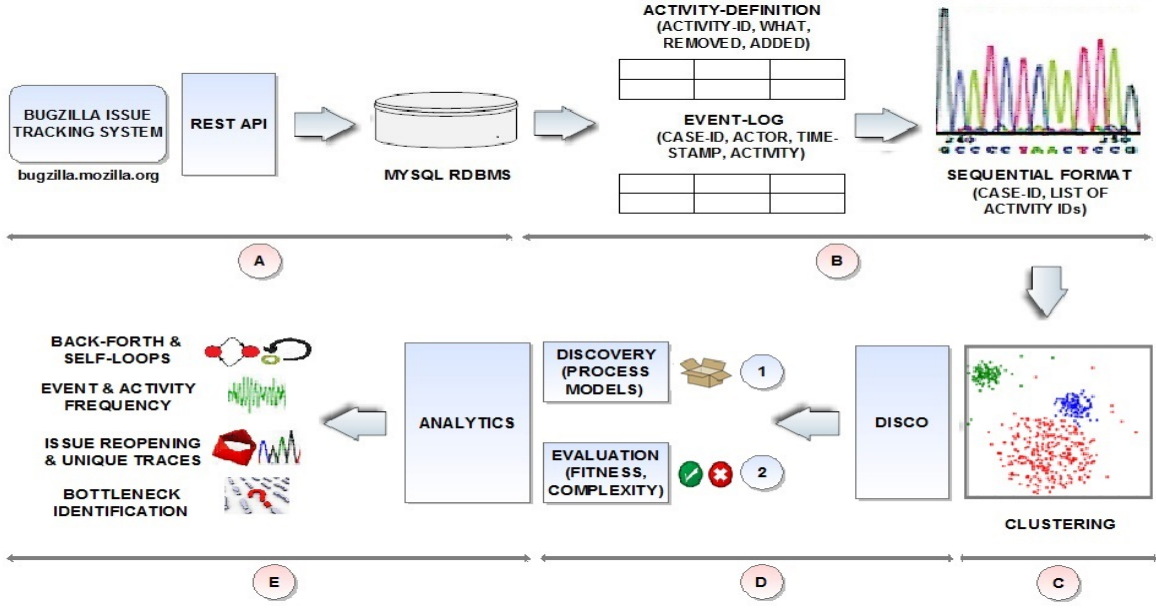


Fig. 1: Architecture Diagram and Data Processing Pipeline for Anvaya Framework (Clustering-Based Approach for Improving the Goodness of Software Process Models Derived from Event-Logs).

II. RESEACH FRAMEWORK AND SOLUTION APPROACH

Figure 1 shows the architecture diagram and the 4 step data processing pipeline for the Anvaya Framework. The first step consists of extracting Issue Tracking System (ITS) data for the Firefox project using the Bugzilla REST API (an HTTP version of its XMLRPC and JSONRPC APIs)¹ and saving it in a MySQL Database. We extract the complete history (life-cycle) of all closed bugs. The history consists of five fields: Who, When, What, Removed and Added. An event in an event-log for a process model discovery algorithm requires a minimum of four fields: Case ID (or the Trace ID for the process instance), Actor, Timestamp and Activity. We map the ITS Issue ID as the Case ID, Who as Actor, When as Timestamp and a combination of What, Removed and Added as Activity.

We convert the history into an Event-Log table consisting of three columns [Case Id, Timestamp and Activity] where Activity column consists of the Activity-ID corresponding to What, Added and Removed in the Activity-Definition Table I. We extract, label and output all the unique activities from the Bugzilla history into the Activity-Definition Table I. For labelling, we use a three letter code which reflects and indicates the activities performed. We identify 81 unique activities in our dataset. Due to limited space, we present the count and description of only 11 unique Activity-IDs in Table I. We structure the Event-Log data in increasing order of Case IDs and activities within a case instance in increasing order of timestamp. We transform the data into a sequential format since we are applying sequential data clustering. We adapt the K -medoid algorithm to cluster the

sequential data using two different distance metrics: Longest Common Subsequence (LCS) and Dynamic Time Warping (DTW). Output of this step is a set of k clusters. The clustering algorithms are explained in Section IV. We generate a single process model from the entire event-log data as well as for each cluster obtained in the previous step using a process mining tool Disco² that uses the fuzzy miner algorithm [3]. We choose Disco because of its ability to manage large event logs and produce complex models. We evaluate the goodness of these process models using cyclomatic complexity and fitness metrics. The last step of Anvaya framework is the Analytics Step where we study and mine useful information from the process models generated from the clusters and show benefits of trace clustering in analysis of back-forth & self loops, bug reopening, and bottlenecks.

III. EXPERIMENTAL DATASET

TABLE II: Experimental Dataset Details (Mozilla Firefox Project)

| Attribute | Value |
|---|------------------|
| Project | Firefox |
| First Issue Report Date | 1 January 2013 |
| Last Issue Report Date | 31 December 2013 |
| Data Extraction Date | 16 October 2014 |
| Number of Open Issues | 3399 |
| Number of Closed Issues Used | 11804 |
| Number of Activities in Closed Issues | 81 |
| Number of Events Reported for Closed Bugs | 178331 |

We extract close bug report data for Firefox Browser because closed bugs have completed their lifecycle. We do

¹ https://wiki.mozilla.org/Bugzilla:REST_API

²Disco is a process mining toolkit for which we obtained the academic license.

TABLE I: Count and Description of 11 out of 81 Unique Activities in the Experimental Dataset.

| Activity | Acronym | Count | Description |
|-----------------------------|---------|-------|---|
| Assigned To | ASS | 4274 | Bug is assigned to the resolver by the triager. |
| Carbon Copy | CCC | 48387 | Adding/Removing people in addition to Reporter, Resolver and QA Contact to the CC list of the bug in order to notify them about the bug's progress. |
| Custom Field Blocking | CFB | 573 | Nominating the bug to stop a release by setting the appropriate blocking flag ³ . |
| Is Confirmed | ISC | 1106 | Confirming the bug to be true i.e. issue raised is actually a bug. |
| QA Contact Assigned | QAC | 1271 | Contacting Quality Assurance agent for either confirming the bug or verifying the fix. |
| Status New Resolved | SNR | 4492 | The bug status changes from New where it was processed to Resolved where resolution has been performed and is awaiting verification by Quality Assurance. |
| Status Resolved Reopened | SRR | 702 | The bug status changes from Resolved where its resolution was set to Reopened where the bug is reopened as the resolution is found to be incorrect. |
| Status Resolved Verified | SRV | 731 | The bug status changes from Resolved where its resolution was set to Verified where Quality Assurance has looked at the bug and its resolution and agrees that the appropriate resolution has been performed. |
| Status Unconfirmed Resolved | SUR | 5334 | The bug status changes from Unconfirmed where it was validated whether the bug is true to Resolved where resolution has been set. |
| Summary Modified | SUM | 2362 | The short sentences describing what the bug is about are added/removed. |
| Target Milestone Defined | TAR | 3787 | Setting the milestone field while the bug is open to indicate the release for which the fix is planned. |

not analyse open bug report data because such bugs are still in the pipeline, work is being done on them, and we don't know what shape they are going to take. The lifecycle of a bug consists of several stages. The initial status of the bug is either New or Unconfirmed. From any of these two states it can either go to Assigned state where it is assigned to a resolver by the triager or can be directly Resolved. A bug can have seven resolutions: Wontfix, Workforme, Invalid, Fixed, Remind, Duplicate and Later³. Here onwards, the bug is often Verified and Closed or can be Reopened. A bug is said to be closed if its status has been set to either Verified or Resolved. Table II shows the experimental dataset details for the Mozilla Firefox project. We conduct experiments on publicly available dataset so that our approach or results can be replicated and used for benchmarking and comparison. We share our dataset and associated files by creating a public repository on GitHub⁴

IV. CLUSTERING

We adapt the K -medoid clustering algorithm [4] [5] to cluster the sequential traces using two different distance metrics. The first distance metric that can be used to compute the similarity between two traces is the Longest Common Subsequence metric (LCS Similarity) [6] [7] [8]. Since each trace can be viewed as a sequence of characters, we use the LCS algorithm to determine the length of the longest common sequence of characters which need not be consecutive but follow a left to right ordering. Longer the length of LCS,

more similar will be the traces. The second distance metric we use is Dynamic Time Warping (DTW Similarity) [9] [10] which is used to find similarity between sequences that are structurally similar but can be on a different timescale. Let two sequences be $S1$ and $S2$. Warping path consists of index pairs (i,j) if DTW associates $S1[i]$ with $S2[j]$. This path is subjected to certain restrictions namely, monotonicity, continuity and boundary condition [11]. Out of the many warping paths, an optimal warping path is the one that minimizes the total cost [11]. Warping distance is the summation of element wise distance between $S1[i]$ and $S2[j]$ over all pairs of (i,j) present in the optimal warping path⁵. We assign a cost (distance) 0 if $S1[i]=S2[j]$, otherwise 1 is assigned. Because of such cost assignment, lower the warping distance, more similar are the traces. So, in k -medoid algorithm a non medoid trace is associated to a medoid with highest LCS similarity or lowest DTW similarity. Algorithm 1 describes the steps to compute k clusters using our proposed technique.

V. PROCESS DISCOVERY & EVALUATION

We discover process models from the entire event log as well as the event log of the clusters using Disco. A node in the process model obtained from Disco represents an Activity while an edge represents transition from one activity to another. The process model has a starting node (represented by a triangle symbol), end node (represented by a stop symbol) and activity nodes containing the name and absolute frequency of the activity. Dashed arrows point to activities that occur at

³<https://bugzilla.mozilla.org/page.cgi?id=fields.html>

⁴<https://github.com/ashishsureka/anvaya>

⁵<http://cs.bc.edu/~alvarez/Algorithms/Notes/dtw.html>

Algorithm 1: k Medoid Clustering

Data: Event log in sequential data format
Result: k clusters

- 1 input the value of number of clusters to be formed k .
- 2 read the input event log
- 3 randomly select k traces as initial medoids.
- 4 **foreach** non medoid trace t_i **do**
- 5 **foreach** medoid trace m_i **do**
- 6 calculate similarity score of t_i and m_i using LCS
 Similarity lcs_i or DTW Similarity dtw_i
- 7 assign t_i to m_i with highest lcs_i or lowest dtw_i .
- 8 **foreach** medoid trace m **do**
- 9 **foreach** non medoid trace o **do**
- 10 swap m and o
- 11 compute the total similarity score (cost) of the
 configuration using either lcs_i or dtw_i
- 12 select the configuration with the highest cost while using
 LCS Similarity and lowest cost while using DTW
 Similarity.
- 13 Steps 4 to 12 are repeated till there is no change in the
 medoids

the very beginning or very end of the processes. Transitions between activities are represented by solid directed arrows with the absolute frequency value written over them. The color of nodes and thickness of edges is proportional to their frequency. Darker shade and larger thickness signifies a higher frequency count. Figure 2a shows a process model generated from Disco.

We evaluate the goodness of process models using two metrics defined in the field of process mining, namely complexity and fitness. Process models discovered from clusters should exhibit low degree of structural complexity and high-degree of fitness.

A. Complexity

Complexity has unwanted effects on understandability, comprehensibility and correctness of process models [12]. Out of the many complexity metrics proposed in literature, we use McCabe's cyclomatic number which represents the total number of independent paths possible in the process model [13]. The pseudocode to determine the cyclomatic number of process models obtained from Disco is given in Algorithm 2. The Xml format input of the process model is needed as it carries all the relevant information namely, the number of edges and nodes which is required for calculating the complexity. The higher the complexity value returned by this algorithm, higher will be number of independent paths and thus more complex will be the model.

B. Fitness

One of the major applications of Process Mining is to determine the gaps between the real world as recorded in the

Algorithm 2: Complexity

Data: Xml format input of the process model
Result: Complexity of the process model

- 1 read number of edges e
- 2 read number of nodes n
- 3 complexity= $e-n+2$

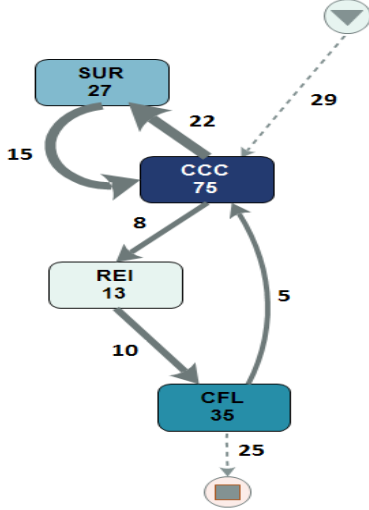
Algorithm 3: Fitness

Data: Xml format input of the process model and Event log in sequential format.
Result: Fitness of the process model.

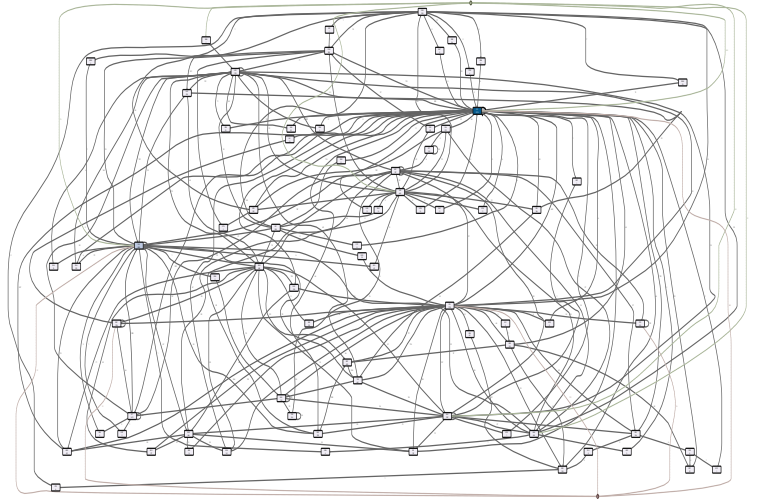
- 1 read Xml format input file.
- 2 **foreach** transition between a source n_i and target node n_j **do**
- 3 adjacency matrix $a_{n_i, n_j} = 1$
- 4 read the input event log
- 5 **foreach** bug id b_i **do**
- 6 add each activity to trace t_i
- 7 **if** t_i is unique **then**
- 8 add it to $unique_traces[]$
- 9 Count its frequency F_i in the event log
- 10 **foreach** entry t_i in $unique_traces[]$ **do**
- 11 $Valid_i = 1$
- 12 $j = 1$
- 13 **while** $j < \text{length of } t_i$ **do**
- 14 **if** $a_{t_i[j], t_i[j+1]} \neq 1$ **then**
- 15 $Valid_i = 0$
- 16 break
- 17 **else**
- 18 $j++$
- 19 **foreach** entry t_i in $unique_traces[]$ **do**
- 20 $\text{FreqValidProduct} = \text{FreqValidProduct} + F_i * Valid_i$
- 21 $\text{FreqSum} = \text{FreqSum} + F_i$
- 22 $\text{Fitness} = \text{FreqValidProduct} / \text{FreqSum}$

event log and the process model⁶. The fitness metric is used to determine the extent to which an event log conforms to the process model generated from that log and vice versa [14]. It can be measured by determining the fraction of traces present in the event log that can be completely replayed by the process model from start to end. The pseudocode to determine the fitness of the process model is given in Algorithm 3 [15]. The fitness of a process model can take any value on a scale of 0 to 1. Fitness value 1 (maximum) indicates that the process model is perfectly aligned with the event log while value 0 (minimum) indicates that the model completely deviates from reality since none of the traces present in the event log are shown in the process model.

⁶http://www.processmining.org/online/conformance_checker



(a) A Process Model example



(b) Main Model

Fig. 2: (a) A Process Model generated from Disco with labels as Absolute Frequency and the arrow thickness & node color proportionate to this frequency. (b) Complex Process Model generated from the the entire event log consisting of 1615 traces.

VI. EXPERIMENTAL RESULTS

To validate the clustering, we apply k -medoid algorithm using LCS and DTW similarity metrics on 1615 process-instances and obtain 6 clusters. Figure 2b shows the complex, spaghetti-like, hard to comprehend process model generated from the entire event log (referred as the main model throughout the paper) obtained from Disco at 100% activity and 12.2% path resolution. The complexity and fitness of main model and all the clusters is shown in Table III. We see that on an average the complexity in a cluster has been reduced by 40.03% and 40.96% while using LCS and DTW metrics respectively clearly showing that clusters are much easier to comprehend and analyse. We notice that process models of 66.67% clusters in case of LCS and 83.34% clusters in case of DTW have a better one to one mapping with the event log and thus show a better fitness value. Throughout our work in further sections, we use LCS distance metric for analysis.

VII. PROCESS MODEL CLUSTER ANALYSIS

In the following section consumable results, actionable information and valuable insights are extracted from all the six clusters obtained using LCS metric. We show that clustering facilitates easier identification of inconsistencies and imperfections and better understanding of the process that would not have been possible by studying the complex spaghetti model.

A. Self-loop Analysis

Study of self-loops is important since such loops indicate intensive problems [16] which are often difficult to detect because it may seem that at each stage some useful work is being done though actually no progress is being made and the bug is just getting transferred [16]. In a process model, a self-loop can be defined as the transition $A \rightarrow A$ i.e. a transition that begins and ends at the same activity. Such anti-patterns

TABLE III: Cyclomatic Complexity along with Percentage Decrease in Complexity of Clusters (DCC) and Fitness Metric of the Spaghetti Model Generated from the entire Event Log as well as the Six Clusters Generated by K -medoid Algorithm using LCS and DTW as the Distance Metrics

| | LCS | | DTW | |
|------------|-----------------------------|---------|-----------------------------|---------|
| | Cyclomatic Complexity (DCC) | Fitness | Cyclomatic Complexity (DCC) | Fitness |
| Main Model | 143 (-) | 0.017 | 143 (-) | 0.017 |
| Cluster 1 | 75 (47.5 %) | 0.178 | 89 (37.7 %) | 0.004 |
| Cluster 2 | 82 (42.6 %) | 0.085 | 93 (34.9 %) | 0.059 |
| Cluster 3 | 106 (25.8 %) | 0.004 | 63 (55.9 %) | 0.328 |
| Cluster 4 | 96 (32.8 %) | 0.070 | 97 (32.1 %) | 0.063 |
| Cluster 5 | 83 (41.9 %) | 0.015 | 78 (45.4 %) | 0.052 |
| Cluster 6 | 72 (49.6 %) | 0.208 | 86 (39.8 %) | 0.078 |

are undesirable and cause redundancy in the bug's trace. Just looking at the count of self-loops of an activity in the event log of spaghetti model is not enough since it might happen that most of these self-loops are occurring in traces of a few bugs in which case we cannot generalize and say that this particular activity causes majority of self-loops in the system. Doing self-loop analysis after clustering similar traces helps us to discover if self-loop of an activity appears only in certain kinds of bugs or if it appears in majority in all the traces. First entry in each cell of Table IV denotes the frequency of self-loop of the activity specified in the first cell of the same row. "-" indicates absence of loop. Due to limited space only some of the activities are represented in Table IV.

1) Self-loop frequency of activity Carbon Copy (CCC) is

TABLE IV: Self Loops and Back-Forth Analysis

| Activity | Main Model | Cluster 1 | Cluster 2 | Cluster 3 | Cluster 4 | Cluster 5 | Cluster 6 |
|----------|---------------|-------------|-------------|---------------|---------------|--------------|------------|
| ASS | 28, CCC/15 | 2, ATT/1 | 5, CCC/1 | 8, CCC/6 | 8, QAC/3 | 5, CCC/7 | -,- |
| ATT | 266, FLA/116 | 24, FLA/6 | 20, FLA/6 | 102, FLA/40 | 48, FLA/18 | 70, FLA/43 | 2, FLA/3 |
| BLO | 152, CCC/39 | 4, CCC/3 | 18, CCC/4 | 72, DEP/18 | 20, CCC/4 | 38, CCC/11 | -, CCC/2 |
| CCC | 6776, FLA/250 | 524, SNR/60 | 875, WHI/37 | 3119, DEP/141 | 1345, SNR/151 | 871, FLA/60 | 42, SUR/53 |
| COM | 2, QAC/3 | -,- | -, QAC/1 | 1, QAC/1 | 1, QAC/1 | -, CCC/1 | -,- |
| DEP | 704, CCC/110 | 9, SNR/2 | 21, CCC/3 | 576, CCC/83 | 41, CCC/6 | 57, CCC/17 | -, CFL/1 |
| FLA | 1612, ATT/464 | 101, CCC/32 | 93, ATT/25 | 614, ATT/168 | 228, ATT/48 | 557, ATT/186 | 19, ATT/12 |
| OPS | 1, PLA/33 | -,- | -, PLA/13 | -, PLA/8 | 1,- | -, PLA/12 | -,- |
| RES | 1, SRU/2 | 1,- | -, SRU/1 | -,- | -,- | -,- | -, SRU/1 |
| SRR | -, RFF/6 | -, RFF/1 | -,- | -, RFF/3 | -, RES/3 | -, RFF/2 | -, RES/1 |
| SUM | 15, CCC/27 | 1, CCC/2 | 3, CCC/1 | 5, CCC/11 | 2, CCC/12 | 4, ASS/1 | -,- |
| TAR | 21, CCC/26 | -, CCC/1 | 4, CCC/1 | 12, CCC/10 | 1, FLA/2 | 4, CCC/11 | -, ASS/1 |
| VER | 6, CCC/20 | 2,- | -, CCC/7 | 1, CCC/7 | -, PRO/1 | 1,- | 2, CCC/5 |

high in all the six clusters with the count being as high as 3119 in Cluster 3. This indicates that many people including users who have no direct role to play in the bug are added in the mailing list. Its an unhealthy practise to repeatedly add/remove people from the mailing list and should be avoided by adding only a few people who are interested in receiving notifications about the bug's progress.

- 2) Many self-loops of activity Attachments (ATT: setting attributes of file related to the bug uploaded by a user) in clusters 3, 4 & 5 indicates that several properties of attachment file⁷ associated with a bug like content-type, description, filename, flags etc keep on changing and attribute fields are not correctly entered by the user while filing the bug.
- 3) Many recurrent loops of Activity FlagTypes (FLA) occur in Clusters 3 and 5. Flags can be of two types: attachment flags and bug flags⁸. Loop involving the former flag indicates that a developer has asked other developers to review his code implying that peer code review practice is followed for quite a lot of bugs while loop involving the latter type indicates that status information of the bug is repeatedly required e.g needinfo flag is set many times sequentially implying that the developer requires more information about the issue raised indicating that bugs are reported with incomplete information.
- 4) High Self-Loop frequency of activity Blocks (BLO) in Cluster 3 indicates that several bugs are repeatedly added in the Blocks field which means a lot of bugs are discovered which depends on the current bug. Bugs in this cluster needs to be resolved on a priority basis as several other bugs are dependent on them.
- 5) Self-loop frequency of activity Depends on (DEP) is extremely high in Cluster 3 indicating that several bugs are identified on which the current bug is dependent. It is interesting to note that self-loop frequency of BLO is also high in this cluster indicating that bugs in these clusters are either dependents or dependees.

B. Back-Forth Analysis

A back-forth loop, also known as ping pong pattern, can be defined as a transition $A \rightarrow B \rightarrow A$ i.e. a transition which begins at activity A, goes to activity B and again ends at A. Second entry in each cell of Table IV contains the activity with which the activity specified in the first cell of the same row is forming a back-forth loop maximum number of times along with the frequency of that loop. An activity A can be in a back-forth loop with multiple activities e.g. $A \rightarrow B \rightarrow A$ with frequency f_1 and $A \rightarrow C \rightarrow A$ with frequency f_2 and $f_2 \geq f_1$. C/f_2 is specified as the second entry in the cell corresponding to Activity A in Table IV. "-,-" indicates absence of loop. Activities forming loops with high frequency can be effectively analysed in clusters. Since bugs with similar life-cycle are clustered together, root cause behind the occurrence of such anti-patterns also becomes easier to identify and study.

- 1) Ping pong patterns that include activity Status Resolved Reopened (SRR) are present in small numbers but are of major interest. The resolve-reopen loop is a problematic pattern. In Clusters 1, 3 and 5 SRR is looping with activity Resolution Fixed (REF) which means that a fixed bug is reopened and again fixed. It happens when the resolution of a resolved bug is found to be incorrect. Such loops are undesirable because the average time to resolve a re-opened bug can be twice as long as the time to resolve a non re-opened bug [17].
- 2) Activity Depends On (DEP) forms a back-forth loop with Carbon Copy (CCC) 83 times and CCC forms a loop with DEP 141 times in Cluster 3 because the teams solving other bugs on which the current bug is dependent need to be informed about the bug's progress so that they can be included in the decision making process of the current bug. Such loops can be reduced by adding just a few people from other teams in the CC list like the team leader instead of all team members.
- 3) Important attributes of the bug like version (VER), operating system (OPS), summary (SUM) and target milestone (TAR) are involved in ping pong patterns indicating that it takes repeated efforts to conclude the values of these fields. Bug reporters should be encouraged to write informative summary of the issue

⁷<https://www.bugzilla.org/docs/3.0/html/api/Bugzilla/Attachment.html>

⁸<http://www.bugzilla.org/docs/2.22/html/flags-overview.html>

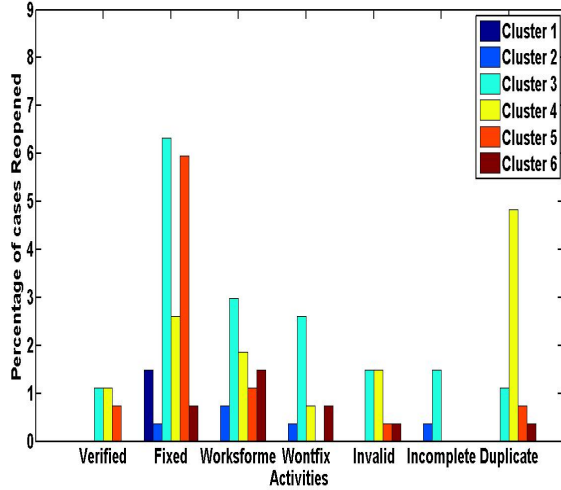


Fig. 3: Histograms showing percentage of cases Reopened for different states (Activities) across the 6 clusters.

and specify fields such as OS and version of the software in which the bug is occurring while filing the bug.

C. Reopen Analysis

Bug reopening refers to the act of changing the status of the bug that was once resolved to Reopened (SRR) as the resolution was found to be incorrect forcing the bug to traverse its lifecycle again. Bug reopening is equally important in open source systems like Bugzilla as it is in closed source or commercial systems [18]. It increases the costs of maintaining the software, lessens the user-perceived quality of the system and leads to extra and needless rework by already loaded developers [17]. Analysis of factors leading to bug reopening helps in improving the quality of bug fixing process and countering all these problems. We take into account the following factors [15] [17] [18] that contribute in reopening of bugs:

- 1) Verified (SRV): A bug verified by a Quality Assurance agent may get reopened if some useful information about the bug becomes available that demands to have it reviewed again.
- 2) Fixed (REF): A fixed bug may have its reopening if the fix proposed seems to have faults and is not complete and entirely correct solution.
- 3) Duplicate (RED): If the bug is not studied deeply and few of its symptoms match with some already existing bug, it is incorrectly assumed to be the case of duplicacy.
- 4) Wontfix (REX)/ Invalid (REN)/ Incomplete (REI)/ Workforme (REW): There are high chances of reopenings if the developer was not able to fix the bug (Wontfix), issue raised was not categorized as a bug (Invalid), bug was reported with incomplete information (Incomplete) or if it was not successfully reproduced (Workforme).

We believe that clustering helps in analysing whether the reopening due to an activity is happening globally throughout the main model or in a certain set of similar bugs.

- 1) Absence of bug re-opening due to Verified (SRV) in Clusters 1, 2 and 6 is supported by the fact that a Quality Assurance agent (QAC) confirms that a proper fix is achieved. While significant re-opening due to Fixed (REF) in all the clusters especially Clusters 3 and 5 indicates bad understanding and management in fixing the bugs in previous releases, leading to loss of time in analysing and correcting the same bug again in the current release (regression bugs). This can be avoided if proper testing and verification of the fix proposed by the developer is done prior to closing the bug.
- 2) Reopening after activity REW is contributed by 5 out of 6 clusters suggesting that re-opening due to Wontfix is occurring globally throughout the dataset and is not limited to some similar types of bugs. Bugs entering into the system are initially difficult to reproduce, thus are left for future references/information using which they will be reopened again. This can be avoided by extracting all possible information about the bug from the reporter to improve understanding before setting its resolution. Also, reporters should be encouraged to describe the bug in as much detail as possible and form for filing a bug should contain various fields that can capture the information about the issue raised in detail.
- 3) Through clustering we are able to segregate those bugs in Cluster 4 which get reopened because of incorrectly getting marked as Duplicate (RED) indicating that the bugs are not properly examined before their resolution is set. Process analyst can analyse such bugs to determine whether the duplicacy is due to similar keywords and title used in describing the bug or if the symptoms of the bugs were not studied deeply leading to failure in identification of the root cause of the issue.
- 4) One reason behind large number of reopenings due to Workforme and Wontfix in Cluster 3 is underestimation of priority of bugs which brings attention to the fact that there is a need to establish clear guidelines and policies to effectively decide priority of a bug.

D. Bottleneck Identification

Bottleneck refers to those areas (activities, transitions, paths) of process model that consume comparatively more time than rest of the system causing the entire process to slow down. Identification of principal factors constraining the process speed can help a process analyst in working upon the causes that deter the performance of a process. We compute the mean time taken for every transition in main model as well as all the clusters. For analysis, we consider transitions taking maximum amount of time and discover severe bottlenecks present in the models.

- 1) Figure 4 shows percentage of bottlenecks taking mean time more than 500 and 1000 days in main model and the 6 clusters. From Figure 4, we observe that percentage

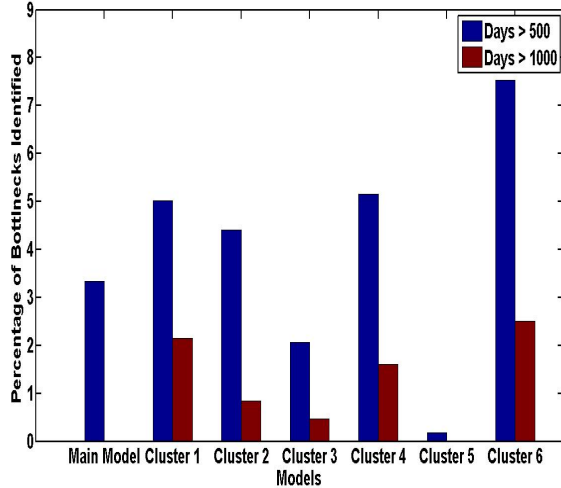


Fig. 4: Histograms showing percentage of Bottlenecks identified in the Main Model and 6 clusters.

of bottlenecks taking more than 500 days (mean value) is greater in Clusters 1, 2, 4 and 6 as compared to the main model. While for duration greater than 1000 days (mean value), each cluster has higher percentage of bottlenecks than the main model. It is due to the absolute count of transitions which is less in a cluster than the main model producing greater mean value for the clusters. Thus, bottlenecks that are not quite evident in the main model are clearly visible in the clusters.

- 2) Set of transitions, taking mean time greater than 1000 days, found in both the main model as well as clusters are:
 - a) $SRV \rightarrow CFB$, $SRV \rightarrow QAC$ implying that after a bug is verified (SRV), there is a large gap before any other actions like contacting Quality Assurance agent (QAC) and setting any blocking flag (CFB) are done. This indicates that once a bug is verified it is not acted upon much.
 - b) $ISC \rightarrow SNR$ suggesting huge delay between the time a bug is confirmed to be true (ISC) to the time appropriate actions are taken to resolve it (SNR) indicating that in some cases it takes a lot of time to understand and confirm that the issue raised is actually a bug. This confirmation step (ISC) can be accelerated to make the system faster.
- 3) The bottlenecks found in clusters (not observed in main model) taking mean time greater than 1000 days are:
 - a) Some of the activities performed before changing the status of bug from New to Resolved (SNR) like ASS and ATT take over 1200 days suggesting large delays in assigning the bug to a resolver (ASS) and studying the associated attachments (ATT). Many a times the attachments are obsolete and their attributes are not defined properly leading

to wastage of time in asking the user to upload the attachment again and resetting their attribute values. Also, delay in assigning the bug to a developer needs to be removed by having a proper procedure to quickly select an appropriate resolver for the bug.

- b) Our analysis shows that setting the resolution to Incomplete, Workforme and Wontfix takes a lot of time as transitions $CCC \rightarrow REI$, $CCC \rightarrow REW$, $CCC \rightarrow REX$ are taking more than 3 years. Higher efficiency is required to identify such cases so that their resolution can be set quickly. Reopening of bugs with these resolutions also takes considerable amount of time indicating that reasons of reopening due to these factors need to be studied in detail with higher priority.
- c) Changing the status of bug from Unconfirmed to Resolved is taking 4 years (indicated by transitions $SUM \rightarrow SUR$, $OPS \rightarrow SUR$) because important attributes of bug like summary (SUM) and operating system (OPS) were not properly defined by the bug reporter, so determining their values took a lot of time.

VIII. DETERMINING THE BEST CLUSTER SOLUTION

Clustering can give many different solutions depending upon the algorithm used, initial cluster centers chosen, number of iterations and number of clusters specified. Out of the many possible solutions, we select the one where clusters have low complexity and high fitness value for enabling better analysis. To test the proposed algorithm, experimental dataset described in Table II is split into four equal sub datasets and each subset is experimented with the proposed algorithm using k -medoid with LCS similarity metric. Algorithm 4 runs the clustering algorithm thrice over the input event log to select the best cluster set. Table V gives the G_Ratio of all the three iterations performed on all four sub datasets as well as the iteration whose solution set is determined to be the best by our proposed algorithm.

IX. RELATED WORK AND RESEARCH CONTRIBUTIONS

Real life event logs are diverse, unstructured and complex leading to formation of 'Spaghetti Models'. The problem of spaghetti process models has been discussed in [19] and [20]. Several techniques have been proposed in literature to cluster traces to deal with complex process models. Bose et al. propose a context aware approach to cluster process instances based on Levenshtein distance [21]. In the technique substitution, insertion and deletion costs of symbols are derived for similarity. The authors evaluate the proposed algorithm on the telephone repair process event log and show that the approach is able to generate clusters with high degree of fitness and comprehensibility when compared to other approaches [21]. In [19] Aalst et al. apply combination of abstraction and clustering techniques to simplify spaghetti-like models discovered using process mining techniques from unstructured and complicated

Algorithm 4: Automate Clustering**Data:** History data of bugs**Result:** Best cluster set

```

1 Perform the preprocessing steps and obtain the sequential data format
  from the history of bugs extracted
2 generate 3 cluster sets  $S_1$ ,  $S_2$  and  $S_3$  using  $k$ -Medoid Clustering that
  uses LCS/DTW similarity for input  $k$  value
3 foreach cluster set  $S_i$  consisting of  $m$  clusters do
4   for  $j \leftarrow 1$  to  $m$  do
5      $\perp$  discover process model  $P_j$ 
6     
$$C\_score_i = \sum_{j=1}^m \frac{Complexity(Xml\ format\ input\ of\ P_j) * t_j}{t_j}$$


$$F\_score_i = \sum_{j=1}^m \frac{Fitness(Xml\ format\ input\ of\ P_j,\ Eventlog\ in\ sequential\ format\ of\ cluster\ C_j) * t_j}{t_j}$$


$$G\_Ratio_i = F\_score / C\_score$$

      $\perp$  where  $t_j$  is total traces in event log of cluster  $C_j$ 
7 return the cluster set  $S_i$  with the maximum  $G\_Ratio$ .
```

TABLE V: Automated Clustering Algorithm Analysis

| Dataset | Iteration | Weighted Complexity | Weighted Fitness | G_Ratio | Result |
|---------|-----------|---------------------|------------------|------------------------|----------|
| 1 | 1 | 90.98 | 0.190 | 2.08×10^{-03} | - |
| 1 | 2 | 92.38 | 0.158 | 1.71×10^{-03} | - |
| 1 | 3 | 90.91 | 0.227 | 2.49×10^{-03} | Selected |
| 2 | 1 | 99.43 | 0.275 | 2.08×10^{-03} | - |
| 2 | 2 | 100.99 | 0.205 | 2.7×10^{-03} | Selected |
| 2 | 3 | 105.8 | 0.213 | 2.01×10^{-03} | - |
| 3 | 1 | 92.05 | 0.125 | 1.35×10^{-03} | - |
| 3 | 2 | 91.39 | 0.106 | 1.15×10^{-03} | - |
| 3 | 3 | 93.47 | 0.218 | 2.33×10^{-03} | Selected |
| 4 | 1 | 81.36 | 0.394 | 4.84×10^{-03} | Selected |
| 4 | 2 | 85.57 | 0.270 | 3.15×10^{-03} | - |
| 4 | 3 | 85.40 | 0.211 | 2.47×10^{-03} | - |

processes [19]. They use significance and correlation metrics to simplify the processes by clustering less significant but highly correlated data [19]. Ferreira et al. propose a sequence clustering approach where each cluster is represented by a first-order Markov chain. [22]. Veiga et al. extended this work by using two dummy states (input and output state) with the Markov chain model for depicting the probability for an event to be the first or last in the sequence [20]. They also suggest several preprocessing steps done before clustering to eliminate undesirable events from the event log [20]. Weerd et al. propose a new technique called ActiTraC (active trace clustering) for trace clustering which uses elements of active learning in an unsupervised environment [23]. The proposed algorithm lessens the divergence between the clustering bias

and the evaluation bias and improves the accuracy and complexity of process models [23]. Song et al. [24] propose a trace clustering technique that uses several perspectives of traces such as performance, transition, case and event attributes organised as a feature vector. Conformance measurement done through process mining in business processes has been shown in [14], [25] and [15]. In context to existing work, the paper makes the following novel contributions:

- 1) Improving the goodness (complexity and fitness) of process models by splitting the event-log into homogeneous subsets by clustering structurally similar traces by adapting the the K -Medoid algorithm.
- 2) Use of Longest Common Subsequence (LCS) and Dynamic Time Warping (DTW) distance metrics in the adaptation of K -medoid algorithm.
- 3) Illustrating the benefits of trace clustering in identifying bottlenecks and study of back-forth & self-loops and bug reopening.
- 4) An algorithm to automate clustering that returns the best cluster set for an event log by determining the goodness of process models.
- 5) An in-depth case study on the open source Firefox browser project to investigate the effectiveness of the proposed approach.

X. CONCLUSION

Analysing the results after mining real world unstructured event logs that show adhoc behavior is difficult due to production of complex spaghetti-like process models. Our work is a contribution towards simplifying these complex models by means of clustering so that they can be easily understood by the process analyst. We adapt K -medoid algorithm using two different distance metrics- LCS and DTW to obtain clusters having good intra-class similarity. K -medoid is an efficient clustering algorithm which is insensitive to outliers and noisy data. Goodness of the models increase as fitness and structural complexity is improved making the models easier to comprehend. We demonstrate the effectiveness of our proposed technique by performing a real life case study on Firefox browser project. We successfully show that clustering enables better analysis, making it easier to identify bottlenecks, study reopening of bugs, self & back forth loops. We propose an algorithm that returns the cluster set with highest goodness ratio to automate the clustering process which is effectively tested on four different datasets.

REFERENCES

- [1] W. Poncin, A. Serebrenik, and M. van den Brand, "Process mining software repositories," in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE, 2011, pp. 5–14.
- [2] V. Rubin, C. W. Günther, W. M. Van Der Aalst, E. Kindler, B. F. Van Dongen, and W. Schäfer, "Process mining framework for software processes," in *Software Process Dynamics and Agility*. Springer, 2007, pp. 169–181.

- [3] C. W. Günther and W. M. P. Van Der Aalst, "Fuzzy mining: Adaptive process simplification based on multi-perspective metrics," in *Proceedings of the 5th International Conference on Business Process Management*, ser. BPM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 328–343. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1793114.1793145>
- [4] L. Kaufman and P. Rousseeuw, *Clustering by Means of Medoids*, ser. Reports of the Faculty of Mathematics and Informatics. Faculty of Mathematics and Informatics, 1987. [Online]. Available: <http://books.google.co.in/books?id=HK-4GwAACAAJ>
- [5] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, Inc., 1990. [Online]. Available: <http://books.google.com/books?id=yS0nAQAAIAAJ>
- [6] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, Jan. 1974. [Online]. Available: <http://doi.acm.org/10.1145/321796.321811>
- [7] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, ser. SPIRE '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 39–. [Online]. Available: <http://dl.acm.org/citation.cfm?id=829519.830817>
- [8] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *J. ACM*, vol. 24, no. 4, pp. 664–675, Oct. 1977. [Online]. Available: <http://doi.acm.org/10.1145/322033.322044>
- [9] J. Kruskal and M. Liberman, "The symmetric time-warping problem: from continuous to discrete," 1983.
- [10] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *KDD Workshop*, U. M. Fayyad and R. Uthurusamy, Eds. AAAI Press, 1994, pp. 359–370. [Online]. Available: <http://dblp.uni-trier.de/db/conf/kdd/kdd94.html#BerndtC94>
- [11] G. Gan, C. Ma, and J. Wu, *Data clustering - theory, algorithms, and applications*. SIAM, 2007.
- [12] J. Cardoso, J. Mendling, G. Neumann, and H. A. Reijers, "A discourse on complexity of process models," in *Proceedings of the 2006 International Conference on Business Process Management Workshops*, ser. BPM'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 117–128. [Online]. Available: http://dx.doi.org/10.1007/11837862_13
- [13] T. J. McCabe, "A complexity measure," in *Proceedings of the 2Nd International Conference on Software Engineering*, ser. ICSE '76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 407–. [Online]. Available: <http://dl.acm.org/citation.cfm?id=800253.807712>
- [14] A. Rozinat and W. Aalst, "Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models," in *First International Workshop on Business Process Intelligence (BPI'05)*, M. Castellanos and T. Weijters, Eds., Nancy, France, September 2005, pp. 1–12.
- [15] M. Gupta and A. Sureka, "Nirikshan: Mining bug report history for discovering process maps, inefficiencies and inconsistencies," in *Proceedings of the 7th India Software Engineering Conference*, ser. ISEC '14. New York, NY, USA: ACM, 2014, pp. 1:1–1:10. [Online]. Available: <http://doi.acm.org/10.1145/2590748.2590749>
- [16] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg, "Designing task visualizations to support the coordination of work in software development," in *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, ser. CSCW '06. New York, NY, USA: ACM, 2006, pp. 39–48. [Online]. Available: <http://doi.acm.org/10.1145/1180875.1180883>
- [17] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. ichi Matsumoto, "Studying re-opened bugs in open source software," G. Antoniol and M. Pinzger, Eds. Springer, 2013, vol. 18, pp. 1005–1042.
- [18] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012 SEIP Track)*. IEEE, June 2012. [Online]. Available: <http://research.microsoft.com/apps/pubs/default.aspx?id=159352>
- [19] W. M. P. van der Aalst and C. W. Günther, "Finding structure in unstructured processes: The case for process mining," in *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*, 10-13 July 2007, Bratislava, Slovak Republic, 2007, pp. 3–12. [Online]. Available: <http://doi.ieeecomputersociety.org/10.1109/ACSD.2007.50>
- [20] G. M. Veiga and D. R. Ferreira, "Understanding spaghetti models with sequence clustering for ProM," in *Business Process Management Workshops, BPM 2009 International Workshops, Ulm, Germany, September 7, 2009. Revised Papers*, ser. Lecture Notes in Business Information Processing, vol. 43. Springer, 2010, pp. 92–103.
- [21] R. P. J. C. Bose and W. M. P. van der Aalst, "Context aware trace clustering: Towards improving process mining results," in *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*, 2009, pp. 401–412. [Online]. Available: <http://dx.doi.org/10.1137/1.9781611972795.35>
- [22] D. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira, "Approaching process mining with sequence clustering: Experiments and findings," in *Proceedings of the 5th International Conference on Business Process Management*, ser. BPM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 360–374. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1793114.1793147>
- [23] J. D. Weerd, S. K. L. M. vanden Broucke, J. Vanthienen, and B. Baesens, "Active trace clustering for improved process discovery," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2708–2720, 2013. [Online]. Available: <http://doi.ieeecomputersociety.org/10.1109/TKDE.2013.64>
- [24] M. Song, C. W. Günther, and W. M. P. van der Aalst, "Trace clustering in process mining," in *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, 2008, pp. 109–120. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00328-8_11
- [25] W. Aalst, "Business alignment: using process mining as a tool for delta analysis and conformance testing," *Requirements Engineering*, vol. 10, no. 3, pp. 198–211, 2005. [Online]. Available: <http://dx.doi.org/10.1007/s00766-005-0001-x>